Midterm (Computer Algorithm) 95.11.20

1. (5%) The following statements are the merge-sort,

```
MERGE-SORT(A, p, r)
if p < r                                  ▷ Check for base case
  then q ← ⌊(p + r)/2⌋                    ▷ Divide
       MERGE-SORT(A, p, q)                ▷ Conquer
       MERGE-SORT(A, q + 1, r)            ▷ Conquer
       MERGE(A, p, q, r)                  ▷ Combine
```

Use the following sequence of 11 data: 4, 7, 2, 6, 1, 4, 7, 3, 5, 2, 6 to illustrate each step of merge-sort.

Ans:

2. (10%) The following statements are the insertion-sort,

```
INSERTION-SORT(A)
for j ← 2 to n
    do key ← A[j]
       ▷ Insert A[j] into the sorted sequence A[1 . . j − 1].
       i ← j − 1
       while i > 0 and A[i] > key
           do A[i + 1] ← A[i]
              i ← i − 1
       A[i + 1] ← key
```

Use the following sequence of 6 data: 5, 2, 4, 6, 1, 3 to illustrate each step of insertion-sort, and analysis the worst-case running time.

Ans:

3. (4 %) What is the value of T(n) for the following recurrence?

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 , \\ T(n-1) + \Theta(n) & \text{if } n > 1 . \end{cases}$$

Ans:

4. (6%) What is the meaning of O(g(n)) , Ω(g(n), Θ(g(n)) for a function f(n)? Explain and draw figures for each definition.

Ans:

5. (6%) Determine the following values for O(g(n)) , Ω(g(n), Θ(g(n)), if g(n) =

(1) $n^2 + n + 6$, (2) $n^2 - 3n - 6$, (3) $n \lg n + 3n$

Ans:

6. (2%) Is $2^{n+1} = O(2^n)$? Why?

Ans:

7. (2%) Is $2^{2n} = O(2^n)$? Why?

Ans:

8. (8%) Determine the T(n) for following recurrences:

- $T(n) = \begin{cases} 1 & \text{if } n = 1, \\ T(n-1) + 1 & \text{if } n > 1. \end{cases}$

Ans:

- $T(n) = \begin{cases} 1 & \text{if } n = 1, \\ 2T(n/2) + n & \text{if } n \geq 1. \end{cases}$

Ans:

- $T(n) = \begin{cases} 0 & \text{if } n = 2, \\ T(\sqrt{n}) + 1 & \text{if } n > 2. \end{cases}$

Ans:

- $T(n) = \begin{cases} 1 & \text{if } n = 1, \\ T(n/3) + T(2n/3) + n & \text{if } n > 1. \end{cases}$

  Solution: $T(n) = \Theta(n \lg n)$.

Ans:

9. (5%)

Let X_{ij} be an indicator random variable for the event where the pair $A[i], A[j]$ for $i < j$ is inverted, i.e., $A[i] > A[j]$. More precisely, we define $X_{ij} = \mathrm{I}\{A[i] > A[j]\}$ for $1 \le i < j \le n$. We have $\Pr\{X_{ij} = 1\} = 1/2$, because given two distinct random numbers, the probability that the first is bigger than the second is $1/2$. By Lemma 5.1, $\mathrm{E}[X_{ij}] = 1/2$.

Let X be the the random variable denoting the total number of inverted pairs in the array, so that

$$X = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij} \ .$$

To find the expect value of X, E[X] =?

Ans:

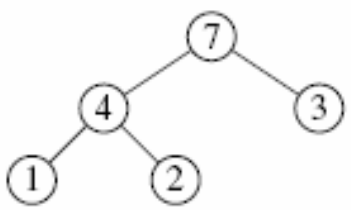
10. (6 %) Use the following example to be a MAX-HEAP tree.

	1	2	3	4	5	6	7	8	9	10
A	4	1	3	2	16	9	10	14	8	7

Ans:

11. (6 %) Following is a heap sort algorithm. Illustrate each step of the heap sort by using the following example.

```
HEAPSORT(A, n)
BUILD-MAX-HEAP(A, n)
for i ← n downto 2
    do exchange A[1] ↔ A[i]
       MAX-HEAPIFY(A, 1, i − 1)
```



Ans:

12. (5%)

Counting sort

Depends on a *key assumption*: numbers to be sorted are integers in $\{0, 1, \ldots, k\}$.

Input: $A[1..n]$, where $A[j] \in \{0, 1, \ldots, k\}$ for $j = 1, 2, \ldots, n$. ARray A and values n and k are given as parameters.

Output: $B[1..n]$, sorted. B is assumed to be already allocated and is given as a parameter.

Auxiliary storage: $C[0..k]$

Is the following counting sort algorithm correct? If it is incorrect, please correct them.

```
COUNTING-SORT(A, B, n, k)
For i ← 0 to k
    Do C[i] = 0
For j ← 1 to n
    Do C[A[j]] ← C[A[j]] – 1
For i ← 1 to n
    Do C[i] = C[i] + C[i +1]
For j ← n downto 2
    Do B[C[A[j]]] ← A[j]
       C[A[j]] ← B[A[j]] -1
```

Ans:

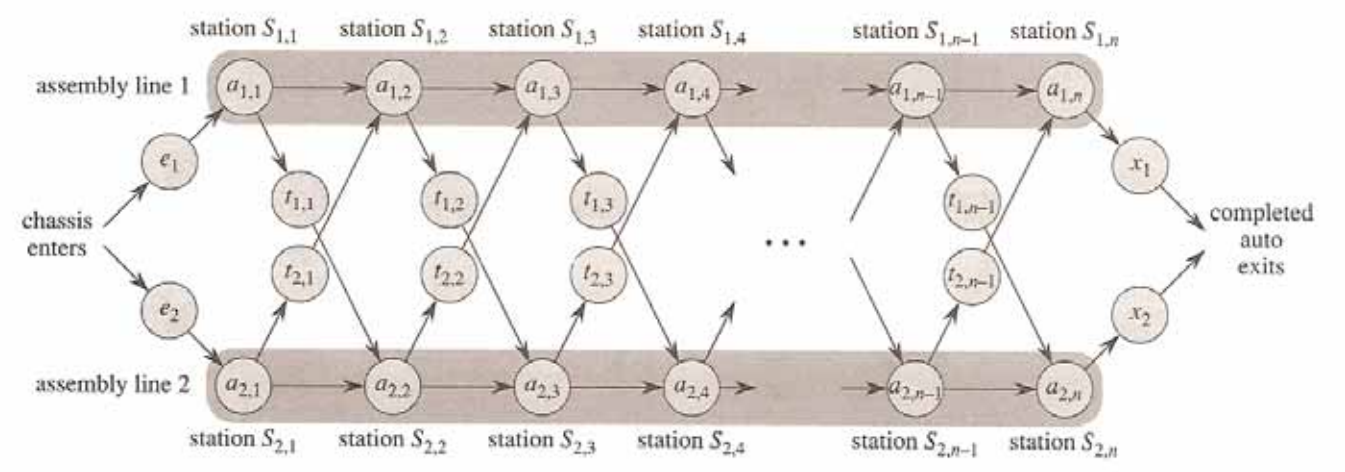
13. (6%) Find a method to find the maximum and minimum with at most $3\lfloor n/2 \rfloor$ comparisons.

Ans:

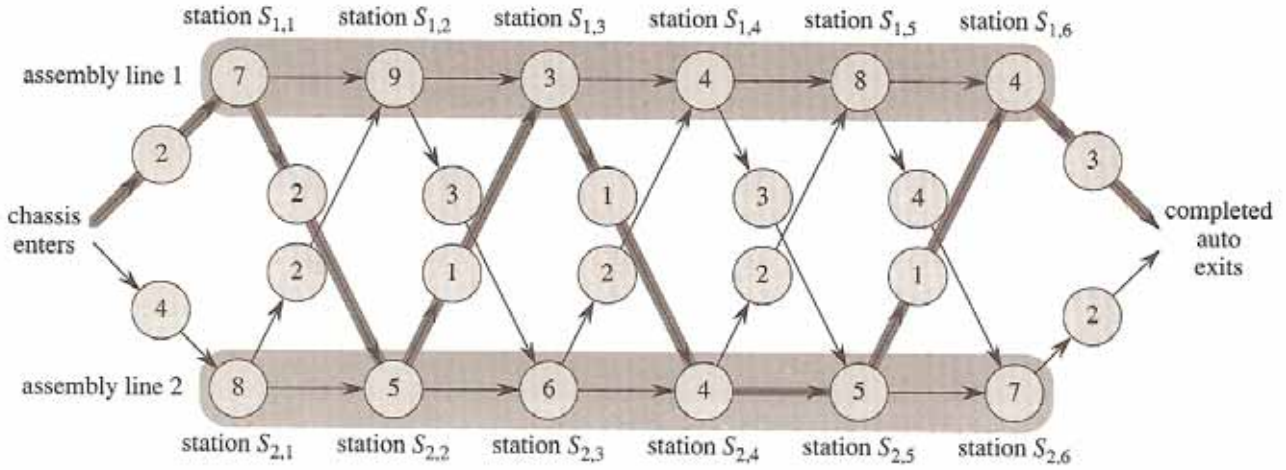
14. (10%) The assembly-line scheduling problem is:

A factor has two assembly lines, shown in the following figure. Each assembly line has n stations, numbered j=1, 2, …, n. We denote the *j*th station on line *i* by $S_{i,j}$. The assembly time required at $S_{i,j}$ by $a_{i,j.}$ The time to transfer a chassis (底盤) away from assembly line *i* after station

$S_{i,j}$ is $t_{i,j}$. We want to minimize the total time through the factor.



For example,



(a)

j	1	2	3	4	5	6
$f_1[j]$	9	18	20	24	32	35
$f_2[j]$	12	16	22	25	30	37

$f^* = 38$

j	2	3	4	5	6
$l_1[j]$	1	2	1	1	2
$l_2[j]$	1	2	1	2	2

$l^* = 1$

(b)

The following is a dynamic programming (DP) method to solve this problem:

Step 1. The structure of the fastest way through the factory

Characterize the structure of an optimal solution. The fastest way through station $S_{1,j}$ is either

- The fastest way through station $S_{1,j-1}$ and then directly through station $S_{1,j}$, or
- The fastest way through station $S_{2,j-1}$ and then transfer from line 2 to line 1, and then through station $S_{1,j}$.

Step 2. A recursive solution

Let $f_i[j]$ denote the fastest possible time to get a chassis from the starting point through station $S_{i,j}$.

The fastest way through the entire factory, we have

$$f^* = \min(f_1[n] + x_1, f_2[n] + x_2).$$

$$f_1[1] = e_1 + a_{1,1},$$
$$f_2[1] = e_2 + a_{2,1}.$$

$$f_1[j] = \min(f_1[j-1] + a_{1,j}, f_2[j-1] + t_{2,j-1} + a_{1,j})$$

for $j = 2, 3, \ldots, n$. Symmetrically, we have

$$f_2[j] = \min(f_2[j-1] + a_{2,j}, f_1[j-1] + t_{1,j-1} + a_{2,j})$$

Combine the above two equations:

$$f_1[j] = \begin{cases} e_1 + a_{1,1} & \text{if } j = 1, \\ \min(f_1[j-1] + a_{1,j}, f_2[j-1] + t_{2,j-1} + a_{1,j}) & \text{if } j \geq 2 \end{cases}$$

$$f_2[j] = \begin{cases} e_2 + a_{2,1} & \text{if } j = 1, \\ \min(f_2[j-1] + a_{2,j}, f_1[j-1] + t_{1,j-1} + a_{2,j}) & \text{if } j \geq 2 \end{cases}$$

Define $l_i[j]$ to be the line number, 1 or 2, whose station j-1 is used through station $S_{i,j}$.

Step 3. Computing the fastest times

```
FASTEST-WAY(a, t, e, x, n)
 1  f1[1] ← e1 + a1,1
 2  f2[1] ← e2 + a2,1
 3  for j ← 2 to n
 4       do if f1[j − 1] + a1,j ≤ f2[j − 1] + t2,j−1 + a1,j
 5            then f1[j] ← f1[j − 1] + a1,j
 6                 l1[j] ← 1
 7            else f1[j] ← f2[j − 1] + t2,j−1 + a1,j
 8                 l1[j] ← 2
 9          if f2[j − 1] + a2,j ≤ f1[j − 1] + t1,j−1 + a2,j
10            then f2[j] ← f2[j − 1] + a2,j
11                 l2[j] ← 2
12            else f2[j] ← f1[j − 1] + t1,j−1 + a2,j
13                 l2[j] ← 1
14  if f1[n] + x1 ≤ f2[n] + x2
15    then f* = f1[n] + x1
16         l* = 1
17    else f* = f2[n] + x2
18         l* = 2
```

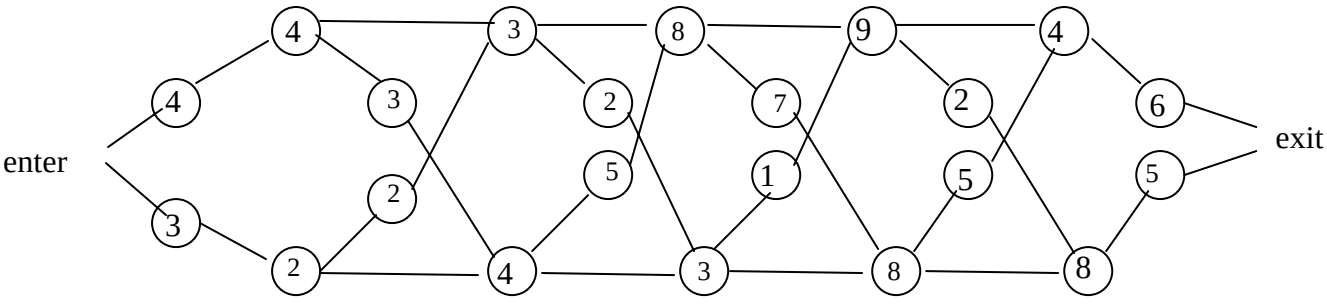
Step 4: Constructing the fastest way through the factory

Print out the stations used in the fastest way.

PRINT--STATIONS (*l*, *n*)

1. *i* = *l**
2. print “line”, *i*, “station”, *n*
3. for j <- n downto 2
4. do *i* <- $l_i[j]$
5. print “line”, *i*, “station”, *j-1*

For the following figure (problem), to compute the values of $f_1(j)$, $f_2(j)$, j=1, .., 5,& f*; $l_1(j)$, $l_2(j)$, j=2, .., 5, & l*.



Ans:

15. (9 %) The following problem is the **optimal binary search trees:**

When searching a key in a binary search tree is one plus the depth of the node containing the key. We want words that occur frequently in the text to be placed nearer the root. It is known as an optimal binary search tree. For each key Ki , we have a probability Pi that a search would be for Ki. A dummy key d_i represents all value between Ki and Ki+1. The following figure is an example:

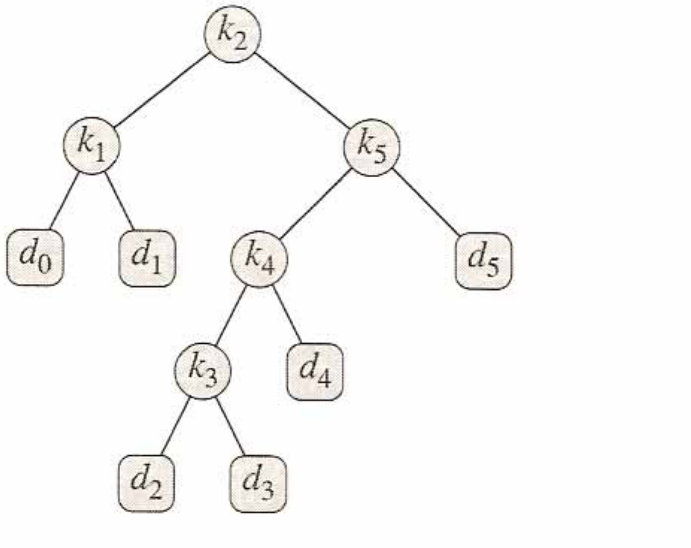


Figure 15.7 Two binary search trees for a set of $n = 5$ keys with the following probabilities:

i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

(a) A binary search tree with expected search cost 2.80. **(b)** A binary search tree with expected search cost 2.75. This tree is optimal.

$$\sum_{i=1}^{n} p_i + \sum_{i=0}^{n} q_i = 1 .$$

$$\begin{aligned} \mathrm{E}[\text{search cost in } T] &= \sum_{i=1}^{n} (\mathrm{depth}_T(k_i) + 1) \cdot p_i + \sum_{i=0}^{n} (\mathrm{depth}_T(d_i) + 1) \cdot q_i \\ &= 1 + \sum_{i=1}^{n} \mathrm{depth}_T(k_i) \cdot p_i + \sum_{i=0}^{n} \mathrm{depth}_T(d_i) \cdot q_i \ , \end{aligned}$$

We want to construct a binary search tree whose expected search cost is smallest. We call such a tree an ***optimal binary search tree***.

Solving it by DP:

Step 1: The structure of an optimal binary search tree.

Consider any subtree, it contains keys Ki, … Kj must also have as its leaves dummy keys d_{i-1}, .., d_j. If an optimal binary search tree T has a subtree T' containing keys Ki, .., Kj, then this subtree T' must be optimal. So, how to select the root of subtree become the issue of finding the optimal binary search tree. Giving keys Ki, .., Kj, and the root Kr, then the left subtree of the root Kr contains the keys Ki, .. Kr-1 (and dummy keys d_{i-1}, .., d_{r-1}), and the right subtree contains the keys Kr+1, .., Kj (and dummy keys d_r, .., d_j). If the root is Ki, then the left subtree only contains the dummy key d_{i-1}(no actual keys). If the root is Kj, then the right subtree only contains the dummy key d_j.

Step 2: A recursive solution.

Let e[i, j] be the expect cost of searching an optimal binary search tree containing the keys Ki, .. Kj. Ultimately, we wish to compute e[1, n]. When j = i-1, we just have the dummy key d_{i-1}, the expected search cost is e[i, i-1] = q_{i-1}. Then the expected search cost of the subtree Ti,j is the sum of probabilities as:

$$w(i, j) = \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l .$$

Thus, if Kr is the root of an optimal subtree containing keys Ki, .., Kj, we have

**** **Question 1.** $e[i, j] = p_r + (e[i, r-1] + e[r+1, j])$ is correct or not? Please correct it and explain the reason, if it is incorrect.

Consider the boundary condition j =i-1.

**** **Question 2.** $e[i, j]$ = ?

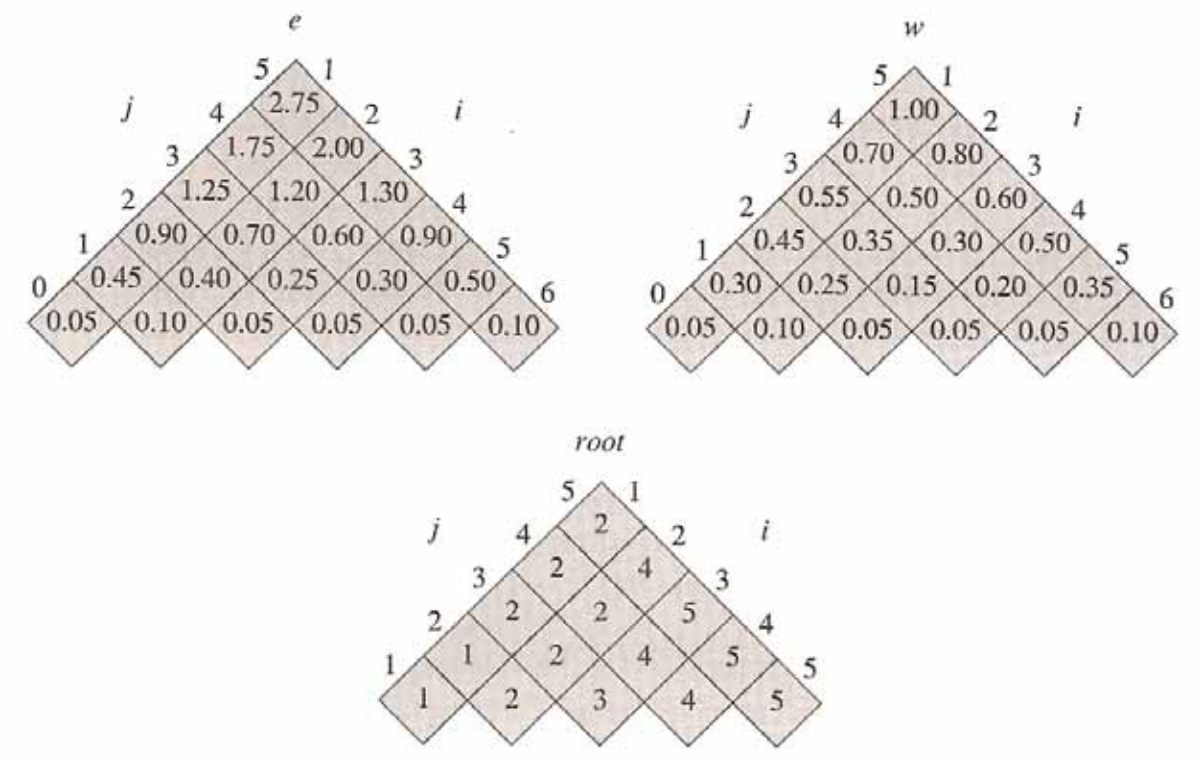
Step 3: Computing the expected search cost of an optimal binary search tree.

We store e[i, j] values in a table e[1.. n+1, 0 ..n]. (e[n+1, n] is used to store the dummy keys d_n , and e[1, 0] is used to store the dummy keys d_0). The pseudocode of the optimal binary search tree is:

```
OPTIMAL-BST(p, q, n)
 1  for i ← 1 to n + 1
 2       do e[i, i − 1] ← q_{i−1}
 3          w[i, i − 1] ← q_{i−1}
 4  for l ← 1 to n
 5       do for i ← 1 to n − l + 1
 6             do j ← i + l − 1
 7                e[i, j] ← ∞
 8                w[i, j] ← w[i, j − 1] + p_j + q_j
 9                for r ← i to j
10                   do t ← e[i, r − 1] + e[r + 1, j] + w[i, j]
11                      if t < e[i, j]
12                         then e[i, j] ← t
13                              root[i, j] ← r
14  return e and root
```

The index l is the width of subtree Ti,j, and r is the root of subtree. For l =1, we compute e[i, i] and w[i, i]. The runnme is $O(n^3)$.

****** Question 3.** Based on the above procedure, if we obtain the following results, please draw the optimal binary search tree.



Ans:

16. (10%) The activity-selection problem is to schedule several competing activities that require exclusive use of a common resource, with a goal of selecting a maximum-size set of mutually compatible activities.

    Suppose we have a set S ={a1, a2, .., an} of n proposed activities that wish to use a resource which can be used by only one activity at a time. Each activity ai has a start time si and a finish time fi, where 0 ≤

si ≤ fi ≤ ∞. If selected activity ai takes place during time interval [si, fi) . Two activities ai and aj are compatible if the intervals [si, fi) and [sj, fj) do not overlap.

The activity-selection problem is to select a maximum-size subset of mutually compatible activities.

For example,

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	8	9	10	11	12	13	14

Then, the largest subset is {a1, a4, a8, a11} or {a2, a4, a9, a11}.

Use the greedy algorithm to solve the following example of activity-selection problem:

$\boldsymbol{i}$	1	2	3	4	5	6	7	8	9	10	11	12
$\boldsymbol{S_i}$	4	1	0	2	3	5	4	7	8	6	12	10
$\boldsymbol{f_i}$	6	3	4	6	7	7	8	13	14	11	15	16

Ans: